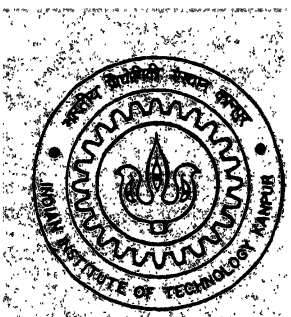
SHIVA: A Network Monitoring Tool and a Study of IITK Computer Center Network

by
Ambarish C. Kenghe



Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur
February, 1998

SHIVA: A Network Monitoring Tool and a Study of IITK Computer Center Network

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Ambarish C. Kenghe

to the
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur
February, 1998

Certificate

Certified that the work contained in the thesis entitled "*SHIVA: A Network Monitoring Tool and a Study of IITK Computer Center Network*", by Mr.*Ambarish C. Kenghe*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Harish Karnick)
Professor,
Computer Science,
IIT Kanpur.

(Dr. D. Manjunath)
Professor,
Electrical Engineering,
IIT Kanpur.

February, 1998

Abstract

With the increasing need for fast and reliable network connectivity, there has been an exponential rise in the number of local area networks. The networks are becoming bigger and more complex. To ensure that networks function efficiently, network monitoring and management is needed to detect any malfunctioning (equipment or software) and take corrective measures. Network monitoring is also useful for observing network performance and traffic patterns, for planning changes and upgrades.

Although there are several other technologies, 10 Mbps Ethernet is by far the most popular technology for local area networks. A general purpose network monitoring tool has been developed and a study of network traffic carried out on a highly populated Ethernet segment at the Indian Institute of Technology, Kanpur (IITK) Computer Center.

Traffic patterns for different protocols were observed and it was found that the network is lightly loaded and that Ethernet bandwidth is not the bottleneck for the kind of applications presently running at the Computer Center. Detailed Graphs have been presented and analyzed. It was found that there is no perceivable correlation between the network delay and the amount of traffic on the Ethernet. Some possible reasons are given for the delayed response times experienced by the users.

Contents

Acknowledgments

It is extremely difficult for me to put into words my gratefulness towards my supervisors Dr. H. Karnick and Dr. D. Manjunath both of whom, despite being extremely busy, always had time for me.

My sincere thanks to Dr. S. K. Agarwal and Dr. R. Tiwari for providing me access to elite facilities at the Computer Center and to Dr. T. V. Prabhakar for helping me out with the color prints.

I must thank Mrs. and Dr. Waghmare for lending me great support during my stay at IIT Kanpur. My thanks to Mrs. and Dr. Karnick for being extremely affectionate in-spite of my rather frequent visits to their residence.

I would like to thank Atul Dobhal, Shyamsundar K. S., Kshitiz Krishna and Nageshwara Rao for helping me in my thesis work. I am also highly indebted to a ll other friends who have been extremely helpful during my thesis work and stay at IIT Kanpur. I take the liberty of omitting the names for the fear of not being able to do justice to all of them.

I have no words to thank my uncle Mr. S. T. Kenghe for supporting me during my M. Tech. Last but not the least, I would like to thank my family members who have sacrificed a lot for my sake.

List of Figures

Chapter 1

Introduction

As more and more computers are getting networked, it is essential to manage and monitor networks in order to ensure that they operate efficiently. An automated tool is essential because, as the size of the network grows, it becomes extremely difficult for the network administrator to rely totally on his intuition and skills to detect and correct problems in the network. And with the current pace of life and work even a short period of network shutdown can be very costly. Network Monitoring is used for finding the traffic patterns and characteristics on a network. Also, network monitoring can be of great help in locating faults and for identifying malicious users on the network.

1.1 Network Management and Monitoring

Network Management is the process of controlling a complex data network so as to maximize its efficiency and productivity. Depending on the capabilities of the network management system, the process will usually include collecting data automatically, processing that data, and then presenting that data to the engineer for use in operating the network. It may also involve analyzing that data and offering solutions and possibly even handling a situation without ever bothering the engineer at all. Further, it will usually include generating reports useful to the network administrator.

To accomplish all the objectives of a network management system the following 5 areas were defined by the International Organization for Standards (ISO) Network Management Forum, as the different functional areas of such a system :

1. Fault Management.
2. Configuration Management.
3. Security Management.
4. Performance Management.
5. Accounting Management.

Network Monitoring deals with the collection of data . It is extremely critical from the network management point of view. The reliability of network management software is a function of its network monitoring module. Simple parameters like throughtput and response time can be of immense help in deciding if some reconfiguration needs to be done.

1.2 Desirable Features

It is desirable that the monitoring software be independent of the Operating System and the interconnecting network. Also, not only should it give the snapshot of network traffic at a given time but should also be capable of collecting data over long periods. The software should be able to detect any problems in the network. The ultimate objective in network monitoring is to build a tool which can correct these flaws 'on-the-fly'.

1.3 Motivation

Network Management software is needed for the smooth functioning of every big or small network. Network monitoring is the most important part of any network management software. Presently, IIT Kanpur has no software which can monitor

and manage the network. There are several monitoring software available in the market and on the Internet. Most of the softwares depend on the operating system and the type of network. There is a need for a portable monitoring software which should generate statistics on the network traffic. The data thus collected has to be properly organized to aid in the detection of problems on a network. This thesis is an attempt to develop a network monitor to aid in detecting faults and to understand the network better. This tool will be extensively used in another project which aims at carrying out trend analysis on the network traffic data.

Most of the network management systems use Simple Network Management Protocol(SNMP). Since the devices on the IIT Kanpur networks (and other old networks) do not support SNMP, the software should not be dependent on SNMP for data gathering.

1.4 Problem Statement

The objective was to develop network monitoring software which can be run on a variety of platforms and networks. The software should record the protocol-wise break up of the network traffic (both number of packets and bytes). It should also record the protocol-wise packet size density (number of packets for a packet size). The software should record the IP-Ethernet mappings and check for any IP clashes on the network. Also, the software should record the network response time for various loads.

Furthermore, the software should be reasonably configurable, should not loose accumulated data on abrupt shutdowns and should be capable of recording host to host, net to host, host to net etc. traffic. Too many packets should not be dropped and the number of dropped packets should be indicated.

1.5 Organization of The Thesis

In Chapter 2 the existing software are described and some background about TCP/IP networking is given. It also discusses some packet capturing mechanisms. Design

and implementation of the software is discussed in chapter 3. Chapter 4 discusses some previous work on this topic and presents the results obtained during this thesis. Conclusions and future work are stated in chapter 5.

Chapter 2

Related Work and Background

As the networks started getting larger, their management became extremely difficult and many software packages were built for managing the networks automatically. The main hindrance in developing good software was to make it general purpose. Software which would work very well on a particular platform and type of network would totally fail to deliver when tried in a different environment. SNMP was developed to cater to this need and, more recently, browser based network management software is being developed. They utilize the portability of Java to make the software highly portable.

Most of the software available in the market is expensive and often difficult to install. Here we discuss some inexpensive network monitoring software available on the web and some of the basics of network monitoring. The next chapter discusses the design of the monitoring tool developed as part of this thesis and Chapter 4 describes how some very simple data can be used to diagnose problems in the network.

2.1 Existing Softwares

Following is a brief description of some of the network monitoring software. Most such software is available for download free of cost. Some of these have been downloaded and used in our monitoring tool.

2.1.1 CNMS

CNMS [DEMK75] stands for Computer Network Monitoring System which was developed by the University of Waterloo Computer Communications Networks Group. This was one the first attempts to monitor computer networks.

CNMS was a special hardware and software system for monitoring the activities of a computer network. In CNMS, each computer to be monitored had to be attached to a monitor. Telephone lines were used to connect the monitors to the controlling computer. Since there was one monitor connected to each computer to be monitored, the system was useful in not only monitoring the network but the system (computers and network) as a whole.

2.1.2 Ping

Ping is the simplest program available. It checks whether a host is reachable and working. Ping uses the ICMP echo-reply mechanism. It sends echo packets to the designated host, if the host sends back reply packets then it is up and running. Ping also records the packet loss and response time. A sample output is given below.

```
PING vidya.ee.iitk.ernet.in (202.141.40.5): 56 data bytes
64 bytes from 202.141.40.5: icmp_seq=0 ttl=254 time=5.7 ms
64 bytes from 202.141.40.5: icmp_seq=1 ttl=254 time=4.2 ms
64 bytes from 202.141.40.5: icmp_seq=2 ttl=254 time=4.1 ms
64 bytes from 202.141.40.5: icmp_seq=3 ttl=254 time=4.2 ms

--- vidya.ee.iitk.ernet.in ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 4.1/4.5/5.7 ms
```

2.1.3 Tcpdump

Tcpdump [Grob] is loosely based on SMI's "etherfind". It was originally written by Van Jacobson as part of an ongoing research project to investigate and improve TCP and Internet gateway performance. The parts of the program originally taken

from Sun's etherfind were later re-written by Steven McCanne of Lawrence Berkeley National Laboratory.

Tcpdump prints out the headers of packets on a network interface that match the boolean expression taken as a command line parameter. In the expression one can specify type (host, net or port), direction (src, dst) and protocol. These basic primitives can be combined by logical *or*, *and* and *not* to define the kind of packets to be captured. By default it displays the headers on screen. There are options to dump them in a file. There are several other options. Given below is an example output.

```
reply ok 120 lookup [|nfs] (DF)
12:48:49.240000 csealpha1.cse.iitk.ernet.in.3272513495 > cseultra1.cse.iitk.ernet.in.n
156 statfs [|nfs]
12:48:49.270000 csex4.cse.iitk.ernet.in.6000 > csealpha3.cse.iitk.ernet.in.2571: . ack
win 4096
12:48:49.280000 cseultra1.cse.iitk.ernet.in.nfs > csealpha1.cse.iitk.ernet.in.32725134
reply ok 1104 statfs [|nfs] (DF)
12:48:49.280000 pc47.cse.iitk.ernet.in.1746 > csealpha1.cse.iitk.ernet.in.domain: 6357
(44)
12:48:49.280000 csealpha1.cse.iitk.ernet.in.3289290711 > cseultra1.cse.iitk.ernet.in.n
148 nop
12:48:49.280000 cseultra1.cse.iitk.ernet.in.nfs > csealpha1.cse.iitk.ernet.in.32892907
reply ok 240 nop (DF)
12:48:49.280000 csealpha1.cse.iitk.ernet.in.domain > pc47.cse.iitk.ernet.in.1746: 6357
1/0/0 (81)
12:48:49.280000 csealpha1.cse.iitk.ernet.in.3306067927 > cseultra1.cse.iitk.ernet.in.n
152 nop
```

A number of other software packages have been developed which provide GUIs and analysis of the data generated by tcpdump.

2.1.4 Netload

Netload [JM] measures the utilization on some network interface and displays a graphical 'stripchart' of the result. The utilization can be for all traffic (the default) or any subset. To specify a subset, one gives a 'filter specification' in the form used by tcpdump. Multiple netload's can be run, all looking at different subsets of the traffic.

The distribution available from LBL [JM] is a very early beta release. This tool was developed to help DARTNet experimenters measure the behavior of network resource management algorithms. This tool requires tcpdump and BPF.

2.1.5 Icmpinfo

Icmpinfo [Dem] is a tool for looking at the icmp messages received on the running host. It displays messages on screen by default but can also be run as a deamon in which case it outputs to SYSLOG. A Sample output generated by *pinging* is given below.

```
icmpinfo: Icmp monitoring in progress...
Jan 28 15:36:09 ICMP_Echo < 144.16.162.172 [pc72.cse.iitk.ernet.in] sz=64(+20)
Jan 28 15:36:10 ICMP_Echo < 144.16.162.172 [pc72.cse.iitk.ernet.in] sz=64(+20)
Jan 28 15:36:11 ICMP_Echo < 144.16.162.172 [pc72.cse.iitk.ernet.in] sz=64(+20)
Jan 28 15:36:12 ICMP_Echo < 144.16.162.172 [pc72.cse.iitk.ernet.in] sz=64(+20)
Jan 28 15:36:13 ICMP_Echo < 144.16.162.172 [pc72.cse.iitk.ernet.in] sz=64(+20)
Jan 28 15:36:14 ICMP_Echo < 144.16.162.172 [pc72.cse.iitk.ernet.in] sz=64(+20)
Jan 28 15:36:15 ICMP_Echo < 144.16.162.172 [pc72.cse.iitk.ernet.in] sz=64(+20)
```

These messages can be very useful in finding out problems due to unreachable hosts, changes in the network etc. and can be used for reconfiguration. The drawback is that one can monitor ICMP messages meant only for the host on which the program is running.

2.1.6 ETHERD

rpc.etherd [GC] gathers statistics about the network as seen from the given network interface. rpc.etherd must be run as root, as it places the interface in promiscuous mode. It requires a front end to display the statistics, such as ethertop or traffic.

The reporting of these statistics is via SUN remote procedure call (RPC). The program works for the Dec MIPS architecture (doesn't work on Alpha).

2.1.7 NFS monitoring

On a campus network NFS traffic can be a major contributor to the overall traffic. Various software have been developed which concentrate on only NFS traffic. Some easily available software are *nfswatch* [CM] and *nfstrace* [Bla92].

NFS monitoring is important because it can help the system administrator in rearranging the resources/users on the network. Sometimes the slow response time on a network is solely due to bad NFS performance.

2.1.8 Netperf

Netperf [Hom] is a benchmark that can be used to measure various aspects of networking performance. Its primary focus is on bulk data transfer and request/response performance using either TCP or UDP and the Berkeley Sockets interface. There are optional tests available to measure the performance of DLPI, Unix Domain Sockets, the Fore ATM API and the HP HiPPI LLA interface.

Netperf is designed around the client-server model. The client and server are called netperf and netserver respectively. The netserver can be either run as a child of inted or as a standalone server listening at a designated port (to be specified by the user). As soon as netperf is started it establishes a control connection to netserver (remote system) and exchanges test configuration information and results. Once the configuration information has been passed, another connection is opened for the measurement itself using the APIs and protocols appropriate for the test.

The most common use of netperf is measuring bulk data performance. This is also referred to as *stream* or *unidirectional stream* performance. Essentially, these

tests measure how fast one system can send data to another and how fast the other system can receive it. The other use of netperf is the measurement of request/response performance. The request/response performance is quoted as *transactions/sec*. A transaction is defined as the exchange of a single request and a single response. From the transaction rate, round-trip average latency can be inferred.

A Windows-NT version of netperf is also available.

2.1.9 Etherman, Interman and Packetman

These tools were developed as part of the ongoing Netman project at Curtin University [SF, Groa]. Etherman and Interman have a very good graphical display but concentrate only on real time traffic. Etherman shows all ethernet traffic while Interman concentrates on the IP traffic only. They show the network connectivity and Packetman has some support for historical data also. They are implemented using packetfilter and NIT (Network Interface Tap) mechanisms.

2.1.10 Watch Dog

The Internet Watch Dog [Ltd] is a commercially available network administration product. Its major function is to display the current status of all the Internet services. The following is a typical report from Watch Dog.

```
Date: Thu, 19 Feb 1970 16:08:40

  Host:Port                          Bad  Good  Delay Best Worst Uptime

  ---------                          ---  ----  ----- ---- ----- ------

  161.29.2.5:SMTP                     23  2440    4.4    0    49     99.07%

  161.29.2.5:POP                          2489    0.1    0    17    100.00%
```

```
161.29.2.5:EMAIL                    5   43    19.1   3  367     89.58%

Times of last five failures:

      Thu 31-Oct 12:59:54 3

      Thu 31-Oct 14:39:57 3

      Thu 31-Oct 14:41:58 3

      Thu 31-Oct 15:47:21 3

      Tue 05-Nov 15:54:43 3
```

Other functions of Watch Dog include the following.

- Reporting of alarm conditions. It reports them through beeps, bringing the Watch Dog window on top, Dialing a specified pager etc.
- Providing Regular Statistics.
- Generating full TCP/IP logs.

The software is available for free trial download.

2.1.11 Netmon

Netmon [Rao97] was developed as part of a M. Tech. thesis at IIT Kanpur. It has two major components : Protocol Analyzer and NFS Analyzer.

Protocol Analyzer receives network data promiscuously through a system and summarizes it into useful statistics, like protocol wise breakup of the traffic, application wise breakup of the traffic, etc. Traffic between various sets of hosts, subnets, specified by the user, is also monitored.

NFS Analyzer deals only with NFS traffic. NFS packets are captured, analyzed, and information like, total number of requests arriving on a file system, both reads and writes, number of requests emanating from a client machine, number of requests issued by a user etc., are gathered. These programs can be run on only DEC OSF. They use the packetfilter mechanism for packet capturing.

As seen in this section there are various software available for network monitoring. The need was for a software which could be run on different systems and collect data for long periods. Since the devices on the IITK network do not support SNMP, the software should not be dependent on SNMP for data collection.

2.2 Hardware Products

Besides the softwares discussed above there are a number of hardware products available for network monitoring. Network monitoring cards [Cor] and network analyzers [CN] are the most common hardware products. These have an advantage of high speed but the analysis and other functionality is limited. Also, collision data can be collected only through hardware devices. Hardware devices on their own are not very popular but may be used in conjunction with software to build a good network management and monitoring system.

2.3 SNMP

With so many network monitoring and management software there was a need for some protocol built specifically for this purpose so that all softwares could use the basic framework built applications on top of it. SNMP [Fei95] (Simple Network Management Protocol) is a simple protocol by which management information for a network element may be inspected or altered by physically remote users.

The SNMP architectural model is best explained in RFC1157, some portion from the RFC is reproduced below.

> SNMP architectural model is a collection of network management stations and network elements. Network management stations execute management applications which monitor and control network elements. Network elements are devices such as hosts, gateways, terminal servers, and the like, which have management agents responsible for performing the network management functions requested by the network management stations. The Simple Network Management Protocol (SNMP) is used to communicate management information between the network management stations and the agents in the network elements.

The configuration, status, and statistical information in a device is made into a database of network management information which is called Management Information Base (MIB). In reality, information may be stored at a device as a combination of switch settings, hardware counters, in-memory variables, in-memory tables, or files. Through SNMP this information can be retrieved like any other information.

SNMP is used by almost all the network management software. It is popular because it can be used to manage network with a wide range of equipment (bridges, repeaters, ASCII terminals etc.) and types of interface technologies (Point-to-Point, DS1, DS3, X.25, Frame Relay, Ethernet, Token-Ring, FDDI, and others). RFC1155, RFC1157, RFC1212, RFC1213 and RFC1271 give a detailed description of the protocol.

2.4 TCP/IP Networking

This and the following few sections discuss networking and packet capturing fundamentals.

The TCP/IP protocol suit [Com95, Fei95] is the single most popular protocol on the Internet. A brief overview of TCP/IP is presented here identifying the major components of the TCP/IP.

2.4.1 Physical Layer

The Physical layer deals with the real world communication hardware. Physical layer standards include detailed description of media, such as coaxial cable, twisted pair, or fiber. They specify physical attachments and signaling methods.

2.4.2 Data Link Layer

The Data Link layer is concerned with moving information between two systems connected by a point-to-point link, a LAN, or a packet network circuit. The Data Link layer is divided into two sub layers - Media Access Control (MAC) and Logical Link Control (LLC). The MAC sub layer defines the rules that must be followed when using a medium. For example, it decides the inter-frame gap, backoff algorithm etc. The LLC sublayer provides connectionless or connection oriented service to the network layer. While providing connection oriented service it takes care of the acknowledgements and retransmissions. The LLC also has a flow control mechanism.

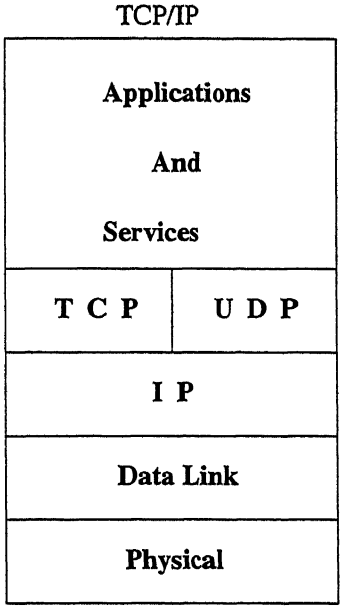


Figure 1: TCP/IP Protocol suite

2.4.3 Network Layer

The Internet Protocol (IP) operates at the Network layer, and has the job of routing data across a network. Any network layer that is based on IP is called an *internet.* Closely connected with IP is the Internet Control Message Protocol (ICMP). ICMP is used to communicate control and error messages linked with the IP.

2.4.4 Transport Layer

Two protocols, TCP and UDP operate at the transport layer in the TCP/IP protocol suite. The TCP (Transmission Control Protocol) is a connection oriented protocol and is responsible for the reliable transmission of data from one application to another. The UDP (User Datagram Protocol) is a connectionless protocol and delivers simple stand alone messages from one application to another.

2.4.5 Applications

Standard TCP/IP applications include electronic mail, file transfer, terminal access, X-server etc.

Any formatted message is called a *Protocol Data Unit* (PDU). The Data Link PDUs are called *frames*, the IP layer PDUs are called *datagrams*, TCP's PDUs are called *segments* and UDP's PDUs are called *user datagrams.*

2.5 Structure of Packets

Packets on the network can be captured by a variety of mechanisms which are discussed later in the chapter. In order to extract useful information from the captured packet one needs to know the exact structure of the packet. This section contains information about the packet structure and headers for TCP/IP suite over ethernet. The packet structure is similar for other protocols with the exception of headers.

An ethernet packet size can vary from a minimum of 64 bytes to a maximum of 1518 bytes (including the checksum of 4 bytes at the end). Ethernet adds a header

of 14 bytes before the PDU that it receives from the network layer. It also appends a 4 byte checksum in the end. It treats the network layer PDU as data. The header contains a 6 byte source address, a 6 byte destination address and a 2 byte type of frame field.

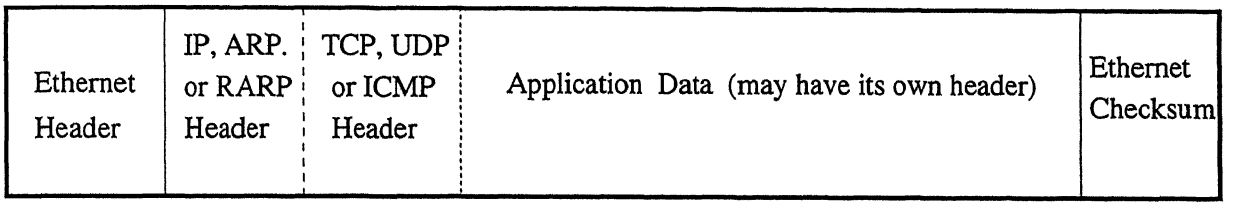


Figure 2: The Ethernet Packet Structure

Next to the ethernet header is the network layer protocol header. The network layer protocol in case of TCP/IP is IP. An IP packet is encapsulated inside an ethernet packet which treats the whole of IP packet as data. Instead of IP there can be other headers such as ARP (Address Resolution Protocol) or RARP (Reverse Address Resolution Protocol) depending on the type of packet. The IP packet encapsulates the Transport layer PDU (UDP or TCP) and the TCP or UDP packet encapsulates the application PDU. A more detailed description of the headers is given in chapter 3 while dealing with the parsing of packets. Figure 2 shows the packet structure of a typical packet on the ethernet.

2.6 Packet Capturing

Packet Capturing is the most basic operation for any monitoring tool. It is very important that the packet capturing mechanism be reliable. In packet capturing reliability means :

1. Packets boundaries should be properly aligned. If the packet boundaries are not aligned properly one may receive a packet which may not contain one packet but portions of two back to back packets or just some portion of a packet.

2. All packets should be captured. If the capturing mechanism is not reliable then one would end up not looking at a lot of packets and hence the data generated would be inaccurate.

Packet Capturing is very much Operating System dependent. For each Operating System the mechanism is different. For Capturing all the packets on a network the Interface Card has to be put into promiscuous mode. All the headers are intact in the packets received through the capturing mechanism. In Ethernet, the checksum is not returned as it is checked by the interface card itself. Since snooping into others packet amounts to breach of privacy, on some systems this facility is available only to the *root* processes. In this section Network Interface Tap (NIT), PacketFilter and Libpcap are described.

2.6.1 Packetfilter

Packetfilter [Cor94] is the mechanism provided by DEC OSF/1 for access to network packets. On DEC OSF/1 systems the access is available to a normal user provided the system administrator has configured the system properly.

The packet filter is a pseudo-device driver which provides a raw interface to Ethernet and similar network data link layers. It is kernel-resident code provided by the DEC OSF/1 Operating System. To the application, the driver appears as a set of character special files (one for each open packetfilter application). The special files can be opened with *pfopen* library routine provided by the Operating System. The packetfilter supports the 10 Mbps Ethernet and FDDI interfaces.

Packetfilter, probably, gets its name from the fact that one can filter packets as desired. For example, only IP packets emanating from a particular host would be captured if so specified. The manual [Cor94] gives a detailed description of packetfilter and its use.

2.6.2 Network Interface TAP

Network Interface Tap (NIT) [Mic89] is the facility provided by Sun machines for packet capturing. It is composed of several STREAMS modules and drivers. These

components collectively provide a facility for link-level network access.

NIT consists of several components that are summarized below.

nit_if This component is a STREAMS device driver that interacts with the systems's Ethernet drivers.

nit_pf This module provides packet_filtering services, allowing uninteresting incoming packets to be discarded with minimal loss of efficiency.

nit_buf This module buffers incoming messages into larger aggregates, thereby reducing the overhead incurred by repeated *reads*.

More detailed description can be found in the manual.

2.6.3 Libpcap

Libpcap [SMJb] is a system-independent interface for user-level packet capture. It provides a portable framework for low-level network monitoring. Apart from being portable the API is also simple to install and use.

The libpcap interface supports a filtering mechanism based on the architecture in the BSD packet filter. Although most packet capture interfaces support in-kernel filtering, libpcap utilizes in-kernel filtering only for Berkeley Packet Filter (BPF) [SMJa] interfaces. On systems that don't have BPF, all packets are read into user-space and the BPF filters are evaluated in the libpcap library, incurring added overhead.

Currently libpcap supports a number of systems and network interfaces. It can be installed on almost all the flavors of unix and supports 10 Mbps Ethernet, 3 Mpbs Ethernet, X.25, Pronet, Chaos, IEEE802.3, ARCnet, PPP, SLIP and FDDI network interfaces. Libpcap is discussed in chapter 3, as the monitoring tool for this thesis was developed using libpcap.

In this Chapter we discussed some existing monitoring software and gave an overview of networking and packet capturing. In the next chapter we discuss the design and implementation of our network monitoring tool, SHIVA.

Chapter 3

Design And Implementation

The network monitoring tool (SHIVA) has three major components - the data collection server, the data interpretation software and the echo server. This chapter discusses the design and implementation of NmON. The discussion is specific to TCP/IP on 10 Mbps Ethernet on which SHIVA is currently implemented.

3.1 Design

It was decided to split the software into the above mentioned three components. This design decision was taken because we would like the data collection server to keep running uninterrupted. So if we want the data collected till a particular time instant we can simply run the data interpretation server and it will produce the desired files which can be viewed with plotting software.

The echo server is executed by the data collection server itself. This section discusses some of the design issues in the development of the three components.

3.1.1 The Data Collection Server

The job of the data collection server is to capture packets, parse them and generate per protocol packet and byte count in a "time slot". A time slot is a period after which the collected data is dumped into a file and the counters are reset. This server also keeps a list of IP-Ethernet mappings and checks for IP clashes. Also,

every hour it dumps the overall packet statistics and the per-protocol packet size distribution. There is provision for host to host, host to net etc. monitoring, also. Data connected with each of this pairing is dumped into a separate file.

First of all, the server reads the IP-Ethernet mapping and packet size distribution from the disk (if at all they exist from a previous monitoring session) and the information about the host to host or net to net monitoring. Now, the interface is put into promiscuous mode so that all the packets can be captured. The filtering mechanism is put into place, if it is requested.

As each packet arrives its length is checked and information extracted from the Ethernet Header is stored. The packet may be an IP, ARP, RARP or some other unidentified protocol.

If the packet is an IP packet then the source and destination IP addresses along with the respective Ethernet addresses are recorded in the IP-Ethernet mapping. Also a check is made for an IP clash. On the Next layer the packet can be ICMP, TCP, UDP or some other transport layer protocol packet. This process of going up the layered structure is continued till all the recognizable headers are processed. Figure 3 gives the protocol dependencies used by the program.

As seen in the figure most of the popular protocols and applications are identified. The unidentified protocols are grouped together as others, at each level. Section 3.3.1 describes how the different protocols are identified.

3.1.2 The Data Interpretation Software

The data interpretation software reads the binary dump files created by the data collection server and creates different ASCII files for each protocol. Even for a particular protocol it creates different files for byte count, packet count and packet size distribution. Since the data stored can be huge therefore it is sampled according to the rate supplied. Also, since the data is dumped at 1 second intervals by default, there is provision for increasing the granularity (say, to 10 seconds) by specifying a higher value through a command line option. The software also shows each file as a plot with the help of *gnuplot*.

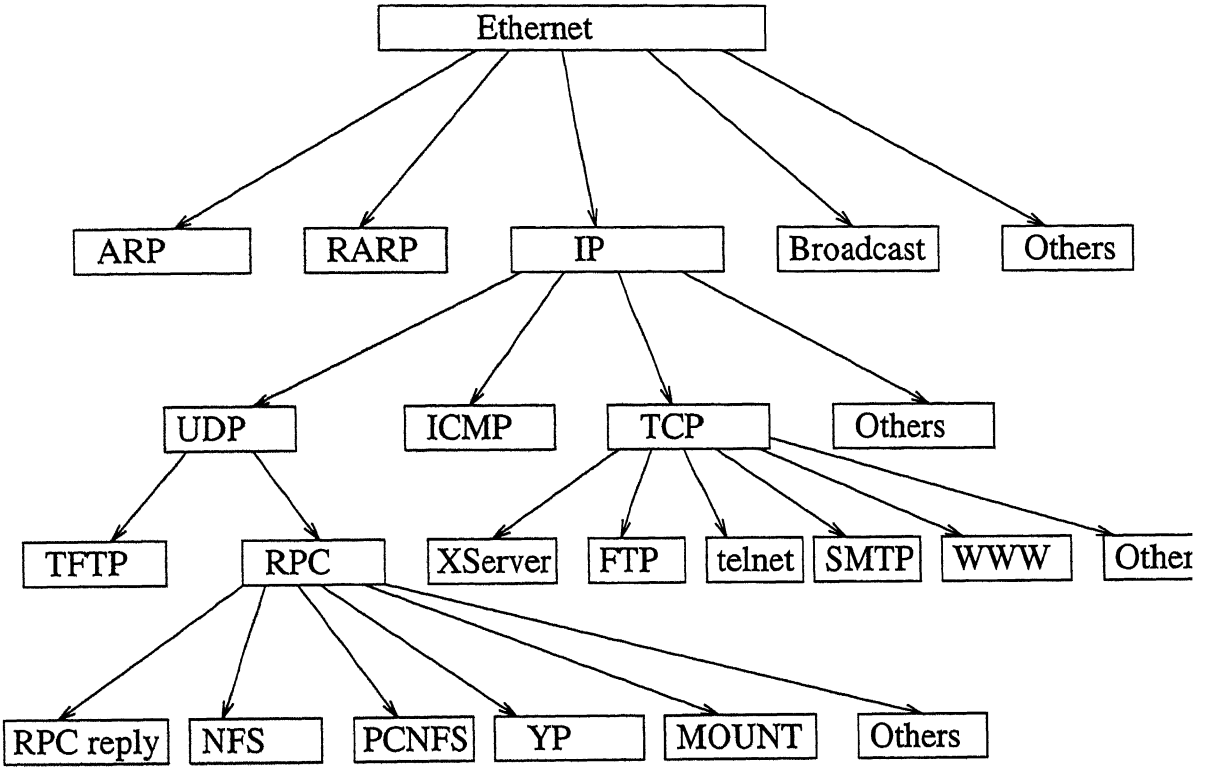


Figure 3: Protocol Dependencies

3.1.3 The echo server

The echo server was built to see how the response time changes with network load, if at all. The purpose here was to exclude the change in response time due to system load. It was decided that the response time be measured by checking the round-trip time of a ICMP echo packet. Therefore the delays associated with buffering at upper layers were avoided. The design of *Ping* program [Ste90] was helpful in designing the echo server.

3.2 Associated Files

Before going into the implementation details let us look at the files associated SHIVA.

slot_dump Binary data on packet and byte counts for all the protocols is dumped

into this file at the end of every time slot.

pack_dist Packet size distribution data (binary) for all the protocols is dumped into this file. The file is overwritten every hour.

ip_clash This file contains the IP clash data (ASCII). The first field contains the IP address and subsequent fields contain the Ethernet addresses with which the IP was associated. The last field contains the time at which the last Ethernet address in the list was associated with the IP. An entry from a real monitoring session is reproduced below.

```
144.16.167.78 0:20:63:0:6:C4: 0:20:63:0:6:2B: 0:20:63:0:5:43: Sun Jan 18 19:02:46
```

inter_pack_arvl This file contains the data for interpacket arrival. The data is refreshed every hour.

ip_clash_dump Whenever an IP clash is caught, the packet is dumped into this file so that later on more information can be had about the IP clash.

ip-ethernet This file contains the IP-Ethernet mapping (binary). This file should be present while starting the server. If no IP-Ethernet mappings are there from any previous monitoring then this would be an empty file.

ip_eth_map This file is an ASCII version of ip-ethernet and is produced by the data interpretation software from ip-ethernet.

pcap_stats The total number of packets received and dropped till the dumping time are recorded in this file, every hour. An example entry is shown below.

```
AT Mon Jan 19 05:19:57 1998
Packets received = 18112324
Packets dropped = 0
```

hosts.monitor This file contains entries for net to net, host to host etc. monitoring. A typical entry is reproduced below. The user's manual gives more info about the format.

```
cd1 1 144.16.167.*
```

This means that separate statistics should be maintained for traffic between cd1 and the network 144.16.167.*.

Other files Apart from the above mentioned files there is a separate ASCII file for each protocol used to store packet/byte count. The file names are the same as used by the CELL structure in section 3.3.1 for variable names. Also, in the 'PACK' directory one separate file is made for storing packet size distribution for each of the protocols. These files are created by the data interpretation program from slot_dump and pack_dist.

3.3 Implementation

Currently the SHIVA has been implemented on Linux in C. The software can be ported on any other operating system with only minor changes. This section discusses the implementation of the software. Some code and data structures were reused from NETMON [Rao97].

3.3.1 The Data Collection Server

This is the major component of SHIVA. At the onset, a startup routine is called which inputs the 'time slot' duration, sets the various signal handlers for different signals, sets the alarm, opens the required files and reads data from the existing files (ip-ethernet and pack_dist). Besides this it also reads the hosts.monitor file and initializes the data structure for storing the host to host, net to net etc. traffic. Then it appends a *0XFFFFFFFF* in the slot_dump file to signify the start of a new monitoring session. Next to the tag it stores the current time.

Next, in the main program, command line arguments are parsed for options and for expression to be used by the packet filtering mechanism. Now the network interface is put into promiscuous mode using the libpcap [SMJb] API. Some of the library calls used in the program are described below. The pcap manual gives more information.

pcap_lookupdev returns a pointer to a network device.

pcap_open_live is used to obtain a packet capture descriptor to look at packets on the network. The maximum number of bytes are defined through this call. The program uses the packet length of 68 bytes to ensure that all the headers are received.

pcap_lookupnet is used to determine the network number and mask associated with the network device found through the pcap_lookupdev call.

pcap_compile is used to compile the filter expression into the filter program.

pcap_setfilter is used to specify the filter program.

pcap_loop is used to collect and process packets. An infinite number of packets are received by specifying the number of packets to be received as -1 (any negative number). The function takes a pointer to a handling function (of specified prototype) as an argument. Whenever a packet is received, this function is called and the packet is passed to it as an argument. The function prototype and a related structure definition is given below.

```
typedef void (*pcap_handler)(u_char *, const struct pcap_pkthdr *, const u_char

struct pcap_pkthdr {
        struct timeval ts;      /* time stamp */
        bpf_u_int32 caplen;     /* length of portion present */
        bpf_u_int32 len;        /* length this packet (off wire) */
};
```

pcap_close is used to close the packet capture descriptor.

pcap_stats fills up the *pcap_stat* structure. The structure is given below.

```
struct pcap_stat {
        u_int ps_recv;          /* number of packets received */
        u_int ps_drop;          /* number of packets dropped */
```

```
        u_int ps_ifdrop;        /* drops by interface XXX not yet
};
```

The packet handler function (called when a packet is received) gets the length of the packet through the *pcap_pkthdr* structure. The packet length found through the structure does not include the 4 byte ethernet checksum. The checksum is verified and removed by the Ethernet card itself and is not part of the packet passed to the handler.

Once the packet is received it is parsed using the protocol dependencies shown in figure 3. Once the header structures are known, packet parsing is a relatively simple task. For each protocol type, a function is defined which takes care of incrementing the different counters, stripping the appropriate header and calling the next relevant protocol routine. The packet handler always calls the *do_ether* function and passes the whole packet to it.

Before going into the parsing mechanism let us look at the structure used to store the different counters in a time slot, CELL. For each specific item monitored, a different CELL is allocated. For example if we choose to monitor the traffic between **cd1** and **cd2** then a separate CELL has to be allocated for storing that statistics. By default one CELL is allocated to all hosts monitoring. The CELL structure is given below for the sake of completeness and future reference.

```
typedef unsigned long COUNTER; /* counter */

typedef struct  {
COUNTER ip_cnt;
COUNTER ip_byte;
COUNTER arp_cnt;
COUNTER arp_byte;
COUNTER rarp_cnt;
COUNTER rarp_byte;
COUNTER nw_others;
```

```
COUNTER nw_others_byte;
} NW_LEVEL;

typedef struct {
COUNTER udp_cnt;
COUNTER udp_byte;
COUNTER tcp_cnt;
COUNTER tcp_byte;
COUNTER icmp_cnt;
COUNTER icmp_byte;
COUNTER tp_others;
COUNTER tp_others_byte;
} TP_LEVEL ;

typedef struct {
COUNTER rpc_call;/* counts only rpc calls */
COUNTER rpc_call_byte;
COUNTER rpc_cnt; /*counts rpc replies*/
COUNTER rpc_byte;
COUNTER nfs_cnt;
COUNTER nfs_byte;
COUNTER telnet_cnt;
COUNTER telnet_byte;
COUNTER ftp_cnt;
COUNTER ftp_byte;
COUNTER smtp_cnt;
COUNTER smtp_byte;
COUNTER www_cnt;
COUNTER www_byte;
COUNTER nntp_cnt;
COUNTER nntp_byte;
```

```
COUNTER xserver_cnt;
COUNTER xserver_byte;
COUNTER tftp_cnt; /* TFTP count */
COUNTER tftp_byte;
COUNTER yp_cnt; /* Yellow pages traffic */
COUNTER yp_byte;
COUNTER mount_cnt; /* NFS mount protocol count */
COUNTER mount_byte;
COUNTER pcnfsd_cnt; /* PC-NFSD traffic count */
COUNTER pcnfsd_byte;
COUNTER tcp_app_others;
COUNTER tcp_app_others_byte;
COUNTER udp_app_others;
COUNTER udp_app_others_byte;
} APP_LEVEL ;

typedef struct {
COUNTER raw_cnt; /* ethernet pkt's raw cnt */
COUNTER raw_byte;
COUNTER bdcst_cnt;
COUNTER bdcst_byte;
NW_LEVEL nw_lvl;
TP_LEVEL tp_lvl;
APP_LEVEL app_lvl;
} CELL ;
```

At the end of every time slot the CELL is dumped into the slot_dump file and the counters are reset. Also, every hour the packet size distribution data and packet statistics are dumped into pack_dist and pcap_stats file respectively.

The *do_ether* function looks at the Ethernet header and stores the source and destination Ethernet addresses, increments the packet/byte counts and then looks up the *type of packet* field to find out the next protocol (IP, RARP or ARP). If the

protocol is not one of these then it takes the packet as *nw_others*. The Ethernet packet structure is defined by the following header.

```
typedef struct ether_packet {
        unsigned char  dhost[6];
        unsigned char  shost[6];
        unsigned short type;
        unsigned char data[MTU];
} ether_packet;
```

If the packet is RARP or ARP, not much processing is needed. If the packet is IP then *do_ip* is called and the packet is passed to the function, ofcourse without the Ethernet header. The following structure (defined in /use/include/netinet/ip.h) is used to define the IP header.

```
struct ip
{
#if defined(__LITTLE_ENDIAN_BITFIELD)
        __u8    ip_hl:4, /* header length */
                ip_v:4; /* version no. */
#else
        __u8    ip_v:4,
                ip_hl:4;
#endif
#define IPVERSION       4
        __u8    ip_tos; /* type of service */
        __u16   ip_len; /* total length of the packet */
        __u16   ip_id; /* identification */
        __u16   ip_off; /* offset, used while fragmenting */
        __u8    ip_ttl; /* time to live */
        __u8    ip_p; /* protocol */
        __u16   ip_sum; /*  checksum */
        struct  in_addr ip_src,ip_dst;
```

```
};
```

The IP source and destination addresses are stored and a check is made for IP clashes. The IP address is checked against the already noted IP addresses (existing in the form of a binary tree). If the IP address does not exist in the tree then it is appended to it along with the Ethernet address. If the IP address already exists and its corresponding Ethernet address does not match the stored Ethernet address then it is an IP clash and the pair is stored in a separate binary tree. The structures used are given below

```
struct ip_to_ethernet{
        unsigned long ip_addr;
        char ether_addr[6];
        struct ip_to_ethernet *left;
        struct ip_to_ethernet *right;
}*ip_ethernet_map;

struct ip_clash{
        unsigned long ip_addr;
        char ether_addr[6][6];/* upto six ethernet addresses for a IP!!*/
        int number; /*no. of ethernet addresses reported for the same IP*/
        struct ip_clash *left;
        struct ip_clash *right;
}*ip_clash_tree;
```

Now, the different counters are incremented and then a check is made for the higher level protocol through the *ip_p* field. The protocol can either be TCP, UDP or ICMP. Otherwise it is put into *tp_others* category.

If the higher level protocol is ICMP then no further processing is required. If it is TCP or UDP then the application level protocols are identified by the port numbers. For example SMTP packets would be headed for TCP port no. 25. This is made possible by the fact that most of the standard applications use fixed ports (and protocol, udp or tcp) for communication. The TCP and UDP header structures used are defined in /usr/include/linux/udp.h and /usr/include/linux/tcp.h respectively.

3.3.2 The Data Interpretation Software

This software first checks the command line arguments which should give the slot_dump file, pack_dist file, ip-ethernet file, sampling rate and granularity. The exact usage is given in the user's manual. At least one of the above files should be supplied to be operated on.

If the pack_dist file has been supplied then it opens the file for reading and opens one file each for the protocols specified in the CELL structure above, with the same name as above. As the packet size distribution information was dumped in a certain order defined by the enumerated type p (given below), the packet size distribution for protocols is retrieved one at a time and put into respective files in ASCII form.

```
enum  p {RAW,BDCST,IP,ARP,RARP, NW_OTHERS , UDP,TCP, ICMP,TP_OTHERS,
RPC_CALL, RPC,NFS, TELNET,FTP,SMTP, WWW, NNTP, XSERVER,TFTP,  YP,
MOUNT,  PCNFSD, TCP_APP_OTHERS, UDP_APP_OTHERS};
```

The protocols are evident from the name. Before generating all these files it creates the PACK directory and changes the working directory to PACK so that the files are made in that directory. Once the complete data is stored in all the files, *gnuplot* is invoked through *popen* library call [Man93]. Each of the ASCII files is plotted on the gnuplot window, one after the other.

If the generation of IP-Ethernet mapping is requested, then the program reads the binary data from the specified ip-ethernet file and stores it in ASCII form in the file ip_eth_map.

If packet/byte count data is requested, the software opens slot_dump file and files for each of the protocols. Now, it reads one CELL at a time and stores the different counts in the respective files. If the first 4 bytes of a CELL are *0XFFFFFFFFF* then it treats the next long word as the time at which next monitoring session started. The time at which current monitoring session ended, is calculated by counting the number of CELLs read since the tag for start of the last monitoring session was encountered. The starting and ending times and duration of each monitoring session is printed on the screen using the above mechanism. Whenever a new monitoring session (tag) is encountered, a negative value is put into the ASCII files so that on

a plot one can easily identify different monitoring sessions. A Procedure similar to that used for plotting packet size distribution, is used to plot packet/byte count.

3.3.3 The Echo Server

As already mentioned the Echo Server was developed to study the dependence of network response time on load in the network. The basic idea is to send an ICMP Echo packet and measure the round-trip time and store it against the network load (raw packet/byte count) at that time (1 sec interval). The program is based on the Ping program [Ste90].

A stripped down version of the Data Collection Server is used for getting the raw packet/byte count. The Echo Server starts this program through the *system* [GNU93] library call. This ensures that there is perfect time synchronization between the two processes.

The software takes host name (to be sent Echo packets) and size of the packets as command line arguments. It sets a timer of 10 seconds (can be changed) using the *setitimer* library call. So that every 10 sec a SIG_ALRM signal is delivered to the process. A Signal handler function is defined which sends ICMP packets to the designated host. The process waits for the SIG_ALRM signal in an infinite loop using *pause*.

When a SIG_ALRM is delivered (every 10 seconds), the signal handler takes over and send an ICMP packet and notes the time of sending. When the reply packet arrives, again the time is noted and the difference between sending and receiving time is put as the response time. After receiving reply packet, the process waits for the next SIG_ALRM.

While sending The ICMP packets the RAW socket is used. *root* permission is required to have access to the RAW socket therefore the program can be run from the *root* UID only. While using the RAW socket, headers have to be constructed by the application program. The following structure was used for constructing ICMP header.

```
struct icmp {
        u_char icmp_type;
```

```
        u_char icmp_code;
        u_short icmp_cksum;
        u_short icmp_id; /* for identification */
        u_short icmp_seq; /* sequence no. */
        char icmp_data[1];
};
```

Standard Ping programs include a time-stamp as data and use it to calculate round-trip time. The round-trip time so calculated includes the delay between getting time from the system and actually sending the packet. The desired response time (network) should exclude this delay therefore instead of time_stamping the packet, time is stored after sending the packet (when *send_to* call returns). This ensures the exclusion of the above mentioned delay, at least from the sending side, in the calculated round-trip time. The response (round-trip) time is stored in an ASCII file.

Chapter 4

Results

SHIVA was used to monitor traffic on the Computer Center (CC) Network at IIT Kanpur over a one week period. This chapter starts with a brief overview of some previous measurement studies of computer networks, followed by a description of the environment for the current study ,the CC network. Finally, the results of the measurements are presented.

4.1 Related Work

This section discusses the major results from some previous measurement studies on networks.

4.1.1 Kleinrock and Opderback, 1977

Kleinrock and Opderback [KO77] were one of the first to carry out measurements of traffic on networks. Their measurements focused on the throughput achievable over ARPANET (world's first large scale experimental packet-switching network). The measurements were carried out at the Network Measurement Center (NMC) at UCLA. They measured throughput for two different flow control procedures and identified limitations that flow control procedures place on the throughput. Although the network technology and the kind of applications were very different at that time, the basic philosophy behind network measurements remains the same.

4.1.2 Shoch and Hupp, 1980

Shoch and Hupp [SH80] measured the actual performance and error characteristics of an 2.94 Mbps Ethernet installation at Xerox PARC. They also showed that under extremely heavy artificially generated load, the system showed stable behavior and that the channel utilization approached 98%. They measured throughput and packet size distribution over 24 hour intervals. Their key observations were as follows.

- The network was lightly loaded. The maximum, minimum and mean loads were 7.9%, 0.2% and 0.8% respectively.
- The traffic was bursty.
- More packets were near the minimum packet length region but most of the traffic (bytes) was due to larger sized packets.
- The error rate was low.
- The channel utilization reached 98% under heavy load (artificially generated).
- The system does not become unstable even under extremely heavy loads.
- The Ethernet collision resolution mechanism is fair.

The study proved that the 2.94 Mbps Ethernet is a robust system and does actually achieve the performance levels far better than those predicted theoretically.

4.1.3 Jain and Routhier, 1986

Jain and Routhier [JR86] presented measurements of inter-arrival times of packets on a ring local area network at the Massachusetts Institute of Technology. Their measurements showed that the arrival processes were neither Poisson nor compound Poisson and they proposed an alternative model for packet arrivals called the *Packet Train* model.

In the Packet Train model, the traffic on the network consists of a number of packet streams between various pair of nodes on the network. Each stream consists

of a number of trains. Each train consists of a number of packets going in either direction between the node pair with which the stream (and the train) is associated. The mean inter-packet gap is large (as compared to the packet transmission time) and random. The mean inter-train time is even larger. They also showed that the Poisson and the compound Poisson arrivals are a special case of the packet train model.

4.1.4 Boggs, Mogul and Kent, 1988

Boggs et al [DRBK88] carried out actual measurements on a 10 Mbps Ethernet and showed that for a wide class of applications, Ethernet is capable of carrying its nominal bandwidth of useful traffic, and allocates the bandwidth fairly. They also discuss how implementations can achieve this performance and suggest ways to avoid some problems that have arisen in existing implementations. They gave the following observations on Ethernet installations.

- The standard Ethernet is an unslotted, 1-persistent, carrier sense multiple-access method with collision detection and binary exponential backoff. This was strongly stated because some analyses assume slotted systems, non-persistent or p-persistent systems, or systems without collision detection.

- Very few Ethernets operate at high loads. Typical loads are well below 50% and are often closer to 5%. Even those networks which show high peak loads usually have bursty sources; most of the time the load is much lower. Thus, unless real-time deadlines are important, the capacity of the Ethernet is almost never the bottleneck.

- The well known myth that Ethernet saturates at an offered load of 37% is not true and that Ethernet is capable of supporting its nominal capacity under realistic conditions.

They collected data on bit rate, standard deviation of bit rate, packet rate, standard deviation of packet rate, transmission delay, standard deviation of delay

and excess delay. Based on the statistics collected, they proposed the guidelines for an Ethernet installation.

- Long cables should not be used. To cover large areas, cables should be broken up with bridges or gateways (not repeaters).
- Too many hosts should not be placed on a single cable.
- Proper collision detection and binary exponential software in interface or host software is essential for good performance.
- Use the largest possible packet size. This keeps the packet count down and reduces the likelihood of collisions.

They concluded that for typical applications, Ethernet is capable of good performance even at high loads. However, they go on to suggest that for very high bit rates and longer networks (upwards of 1 Km), ring topology may be a better option.

4.1.5 Gonsalves and Tobagi, 1988

Gonsalves and Tobagi [GT88] carried out a simulation study of several distributions of stations on a linear bus Ethernet. They showed that the aggregate performance depends on the distribution of stations and that the individual performance varies with the location of the station. Furthermore, they concluded that unbalanced distributions can lead to large performance differences between individual stations and the performance of isolated stations becomes very poor as compared to the average. The study also examines the effects of varying the number of buffers per station and the retransmission algorithm.

4.1.6 Gusella, 1990

Gusella [Gus90] analyzed traffic on a 10 Mbps Ethernet local-area network that connects disk less workstations to file servers in the Computer Science Department of the University of California at Berkeley. The distribution of packet lengths and packet inter arrival times were studied for TCP, Network Disk Protocol and NFS.

The network utilization reported by this study was higher than that reported by Shoch and Hupp and yet the network utilization rarely exceeded 20% over 1 minute intervals. For smaller intervals, the utilization frequently reached 30% and the traffic was reported to be bursty.

It was reported that the packet arrival rate is high (50% of packets are followed within 3 ms) and yet there are very few back to back packets i.e. packets for which the inter-packet time is close to the minimum data-link-layer inter-frame spacing of 9.6 μs for a 10-Mbps Ethernet.

4.2 Present Study

The network monitoring tool was used to monitor a highly populated segment on the IITK Computer Center (CC) Network. The CC network consists of an Ethernet switch and different segments connected to different ports of the switch. Three Silicon Graphics servers are connected to one port, Some HP-9000 and DEC machines are connected to a port and more than 60 Pentium machines (running Linux) and more than 30 HP workstations are connected to another port of the Ethernet switch (see figure 4). Other ports get connections from other parts of the IITK network. The Linux machines run *yp* and file systems from the servers are mounted on all the machines. Users are allocated different file systems based on availability and need.

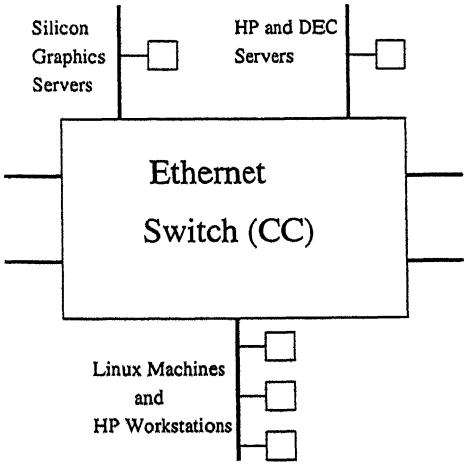


Figure 4: IITK CC Network

The monitor machine was a Pentium (running Linux) connected to the segment on which the CC Linux machines and HP workstations are situated. The segment was monitored for a period of one week. Some key observations and findings are presented in this section.

4.2.1 Inter-packet Arrival Time

Figure 5 shows the distribution of packet arrival times over a period of more than 66 hours. A total of 19,29,4927 packets were received during this period out of which 68,587 were runt packets [1]. No packets were dropped by the kernel. The inter-arrival times are rounded off to the nearest millisecond. The graph has been shown as connected to make it more readable. Inter-arrival times greater than 500 msec have not been plotted in the graph.

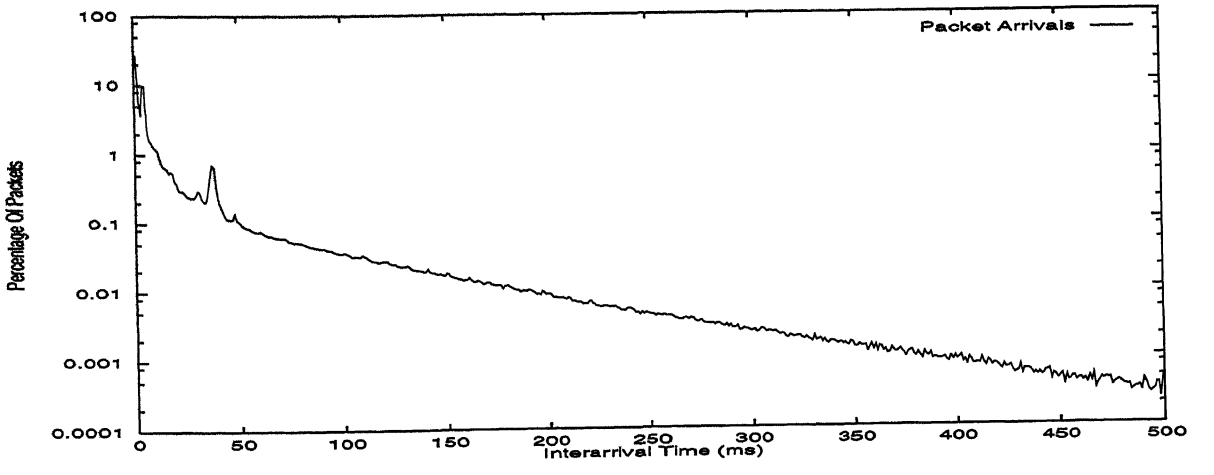


Figure 5: Percentage of Packet Arrivals

As seen in the graph there are quite a few packets having inter-arrival times more than 100 msec and some inter-arrival times are as high as 500 msec. This is in agreement with the observations made by Gusella [Gus90].

[1] Less than 64 bytes in length. This may be a result of collisions

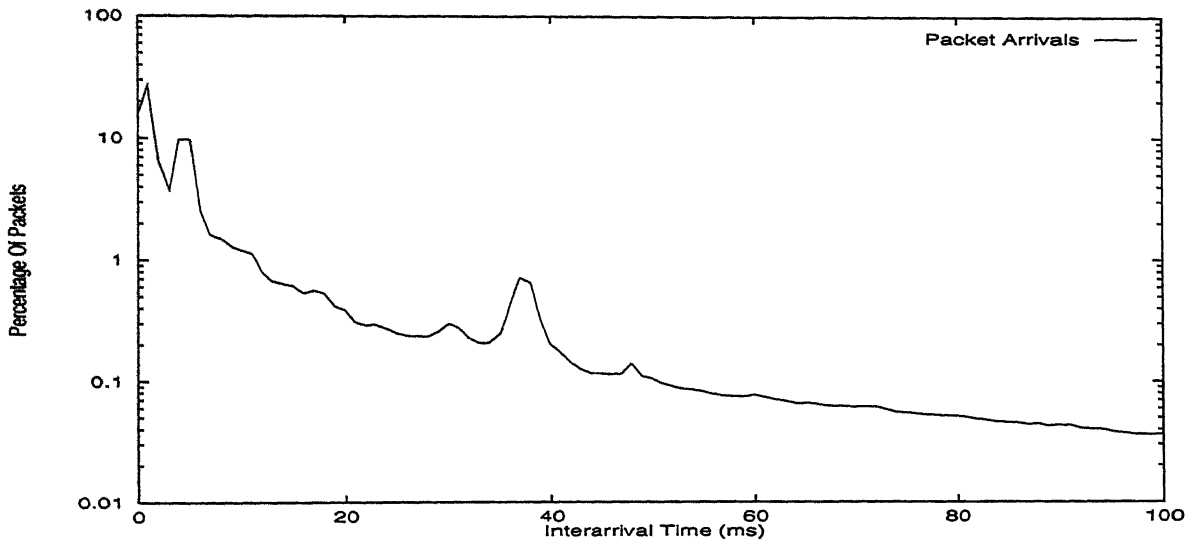


Figure 6: Percentage of Packet Arrivals

Figure 6 takes a closer look at the 0 to 100 msec interval. As already said, there are a high percentage of packets against 0 msec just because the level represents an interval from 0 to 0.5 msec. Still, the percentage of arrivals for 1 msec is almost double the arrivals for 0 msec. The mean inter-arrival time in our measurements was 12 msec. This is a large interval compared to the minimum inter-frame gap of 9.6 μsec for Ethernet.

4.2.2 Distribution Of Packet Size

In this section we discuss the packet size distributions for some predominant protocols. The data was collected for nearly five days on the CC network. The packet size in all the graphs includes all headers and the Ethernet checksum. The minimum and the maximum packet size are 64 and 1518 bytes respectively.

Figure 7 shows the packet size distribution for all the packets. It is seen that most of the packet have sizes between 64 and 174 bytes. There are a large number of packets packets of length 64 bytes. However, most of the bytes transferred through the network are because of larger sized packets (mostly 1518 byte packets). The mean packet size is 97 bytes which is very low as compared to that found in other studies. [SH80, Gus90]. With higher packet sizes higher throughput can be achieved.

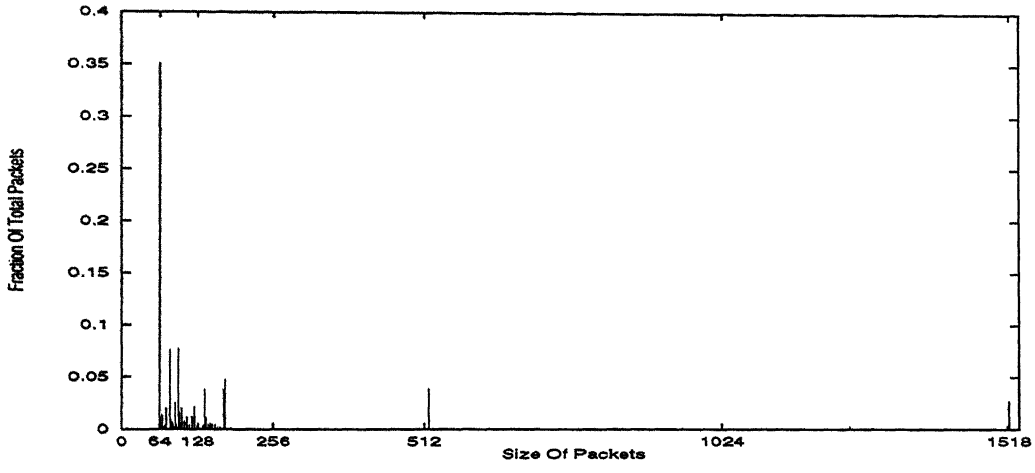


Figure 7: Fraction of all Packets versus Packet Sizes

As will be shown in figure 10 a large fraction of telnet are 64 bytes in length and is a major contributor to lowering the average packet size.

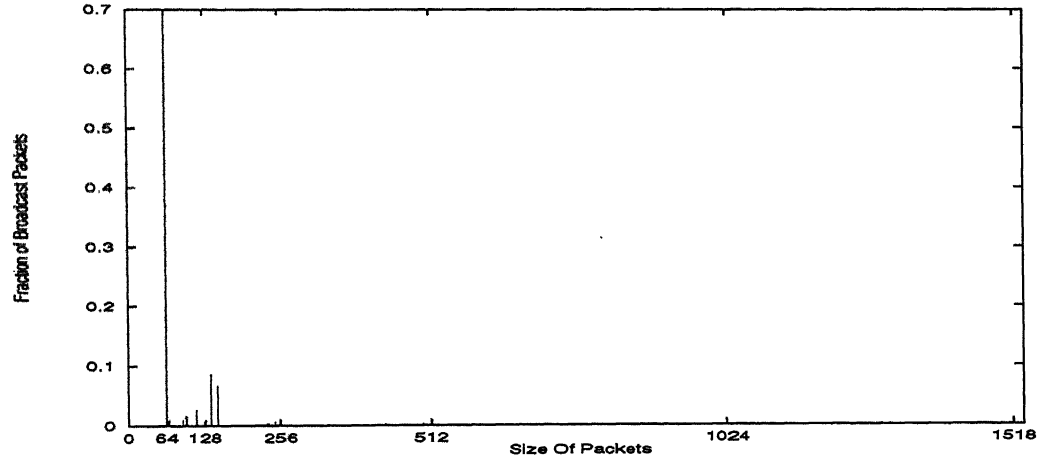


Figure 8: Fraction of Broadcast Packets versus Their Sizes

Almost 70% of the Broadcast packets (Figure 8) are of 64 bytes size. Broadcast packets are generally ARP and BOOTP packets.

Figure 9 shows the packet size distribution for TCP packets. A large number of packets are of size 64 bytes because Telnet (figure 10) uses TCP. Most of the maximum sized packets are due to FTP traffic (figure 12). X-Server traffic (figure 11) constitutes the packets having sizes between 64 and 256 and some of the maximum sized packets.

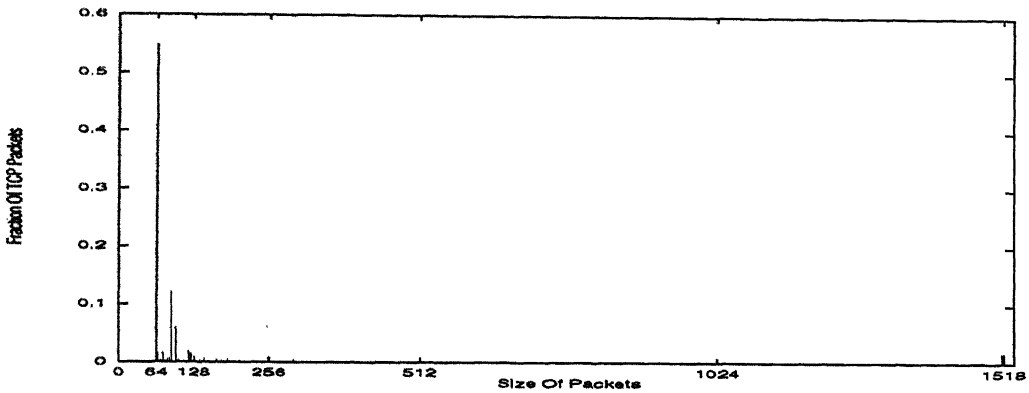


Figure 9: Fraction of TCP Packets versus Their Sizes

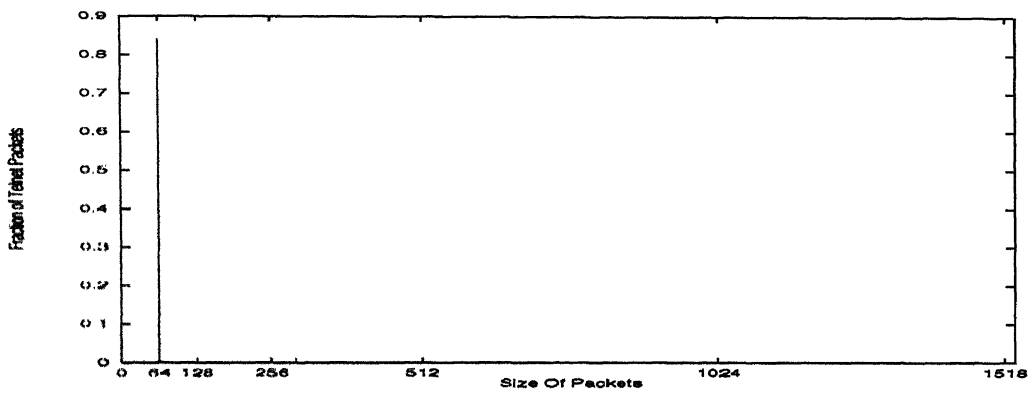


Figure 10: Fraction of Telnet Packets versus Their Sizes

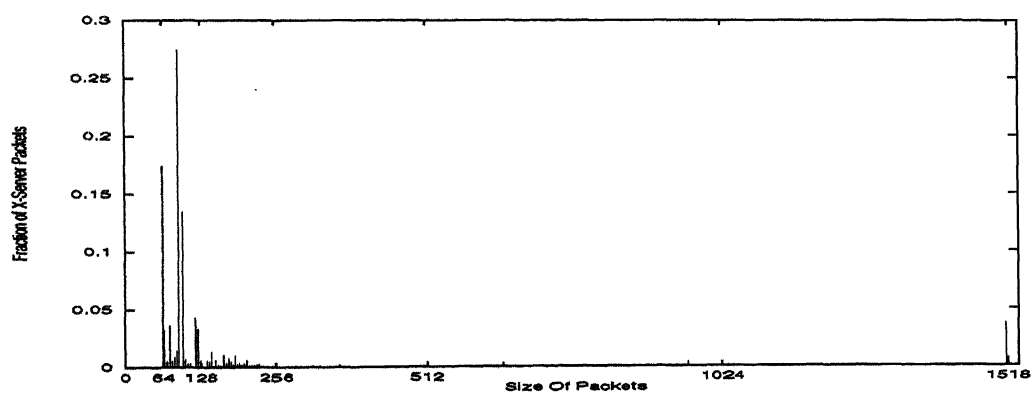


Figure 11: Fraction of X-Server Packets versus Their Sizes

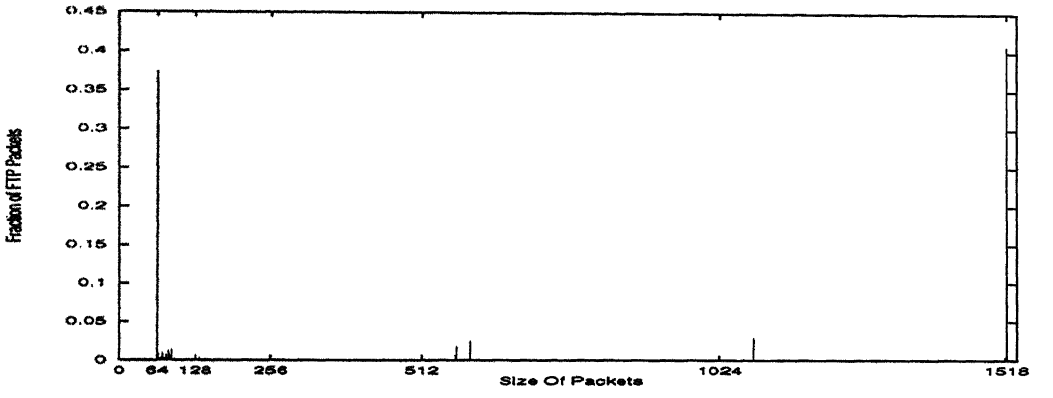


Figure 12: Fraction of FTP Packets versus Their Sizes

The UDP packet (figure 13) sizes are relatively evenly distributed between 64 and 174 bytes. There are some packets of length 1518 bytes almost all of which are due to NFS. The NFS maximum sized packets are generated during file reads and writes.

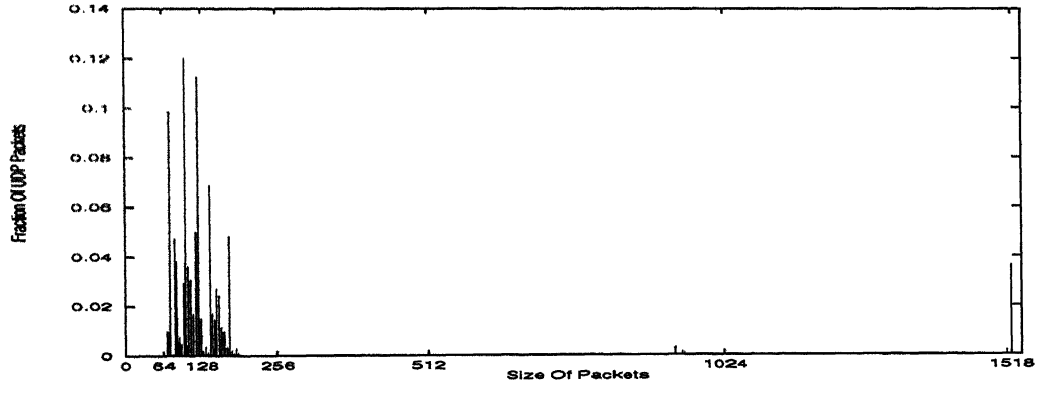


Figure 13: Fraction of UDP Packets versus Their Sizes

Figure 14 shows the packet size distribution of RPC reply packets. The packets of size 1518 are mostly due to NFS file read.

Figure 15 shows the packet size distribution for NFS calls only. The NFS replies are contained in the RPC reply domain. Most of the packets have sizes between 142 and 190 bytes. Quite a few packets have a size of 1518 bytes which are due to file write operation. Some packets are of size 1510 bytes also. They may be the last fragments in a file write operation.

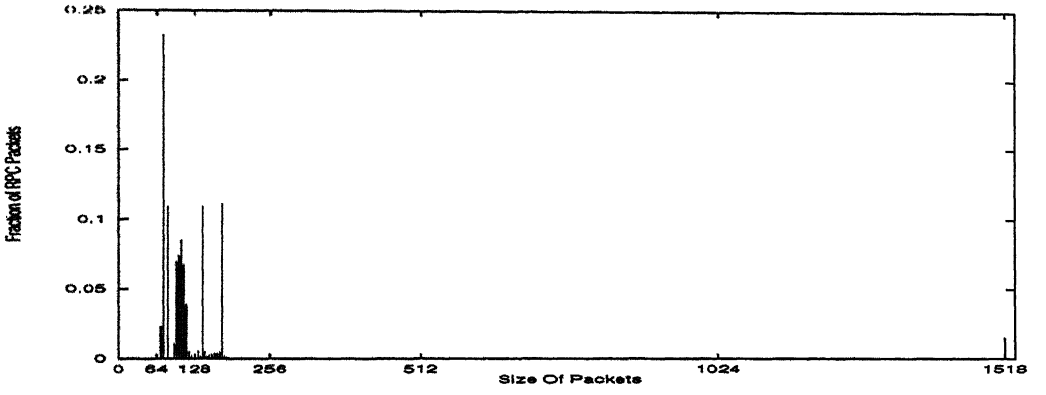


Figure 14: Fraction of RPC Reply Packets versus Their Sizes

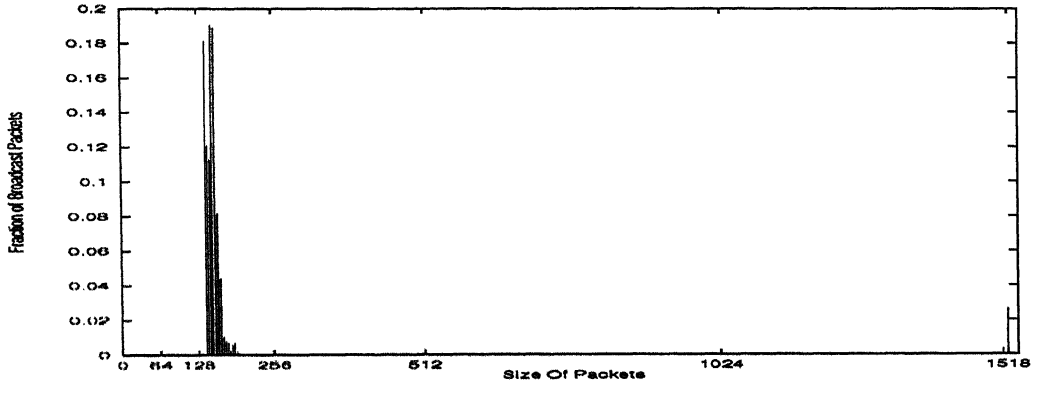


Figure 15: Fraction of NFS (call) Packets versus Their Sizes

4.3 Ethernet Utilization

This section presents the network utilization graphs for various protocols. The measurements were carried out for nearly 5 days (4 days 23 hours) on the CC network. Figure 16 shows the total network utilization averaged over 1 minute intervals.

The mean network utilization is 5.89% which is a reasonably low figure. As seen in the graph, traffic follows a regular pattern depending on the time of the day. The traffic pattern over a 24 hour period on a week day is very much similar to that on any other week day. Towards the weekend the traffic falls drastically. Since the traffic follows a regular pattern over a 24 hour period, subsequent observations have been presented for a period of 24 hours on a week day. The monitoring period starts at 12:43pm on Thursday (see figure 16). Thursday is chosen because the traffic is

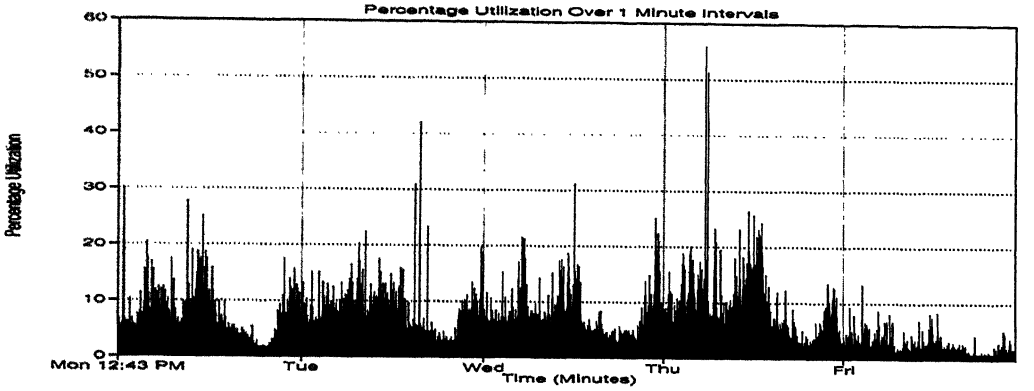


Figure 16: Ethernet Utilization On the CC network Over a Period Of 5 Days

relatively higher on that day.

Figure 17 shows the network utilization over the chosen 24 hour period. The mean utilization is 7.68% in spite of the traffic being a shade higher on the particular day. The mean utilization for the peak period of 2 hours, marked between 'A' and 'B' in the graph, is 15.05%. As expected, there is very little traffic during from 4:00 am to 8:00 am. Figure 18 shows the same utilization graph for 1 second intervals.

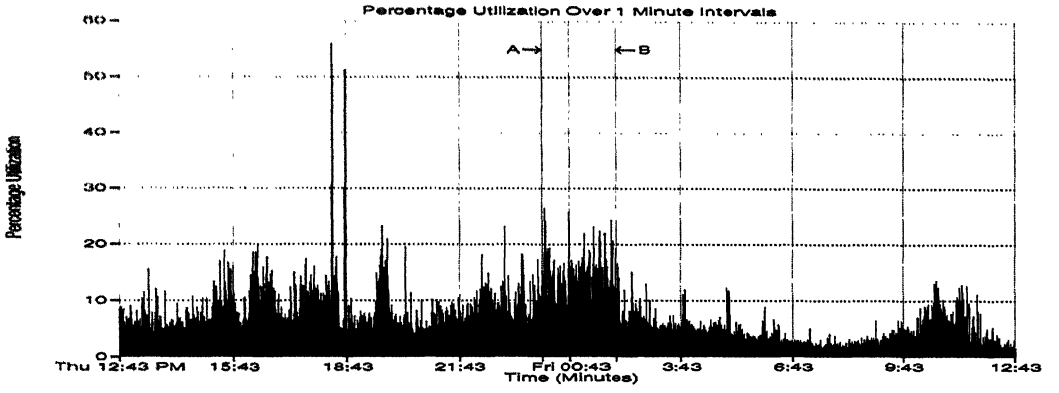


Figure 17: Ethernet Utilization On the CC network Over a Period Of 24 Hours

It is seen that at the 1 second level the network utilization is much more bursty. Inherently network traffic is extremely bursty. The packet train model [JR86] was triggered by this nature of the network traffic. Figure 19 shows the percentage of time the utilization exceeded a certain value. The percentage utilization is taken over 1 second intervals. As can be seen in the graph, even over 1 second intervals,

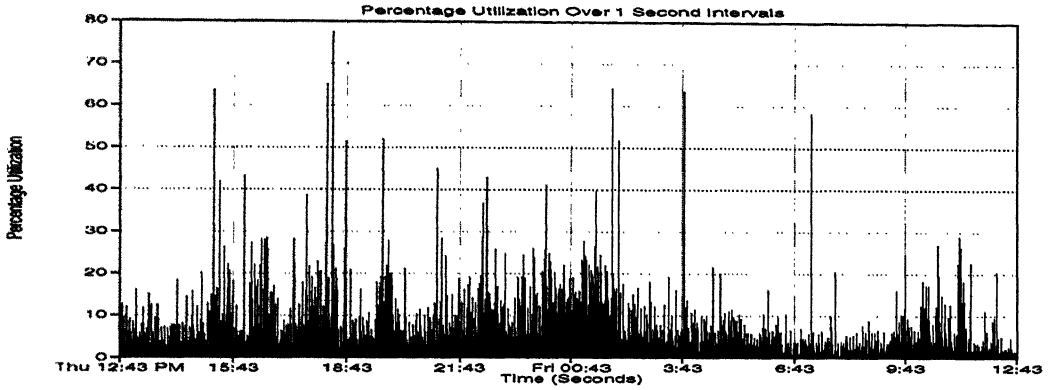


Figure 18: Ethernet Utilization Over a Period Of 24 Hours Sampled At 1 sample/minute

although there are bursts of high utilizations, the number of these intervals is extremely small. In fact, more than 98% of the time the utilization does not exceed 30% in 1 second intervals. Hence, we conjecture that the network is not really the cause of the slow responses that are observed over the CC network.

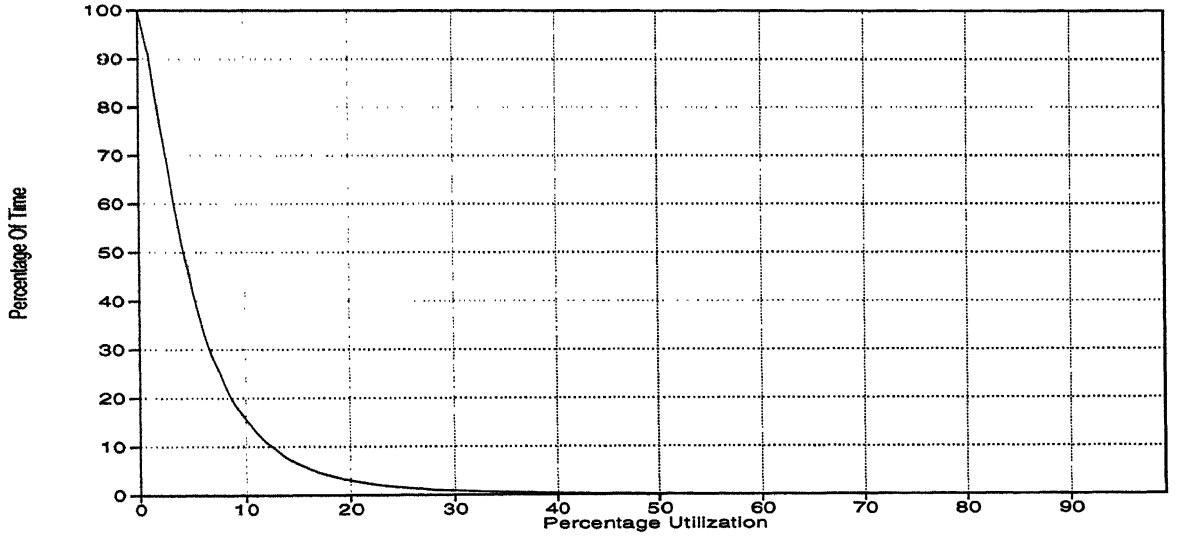
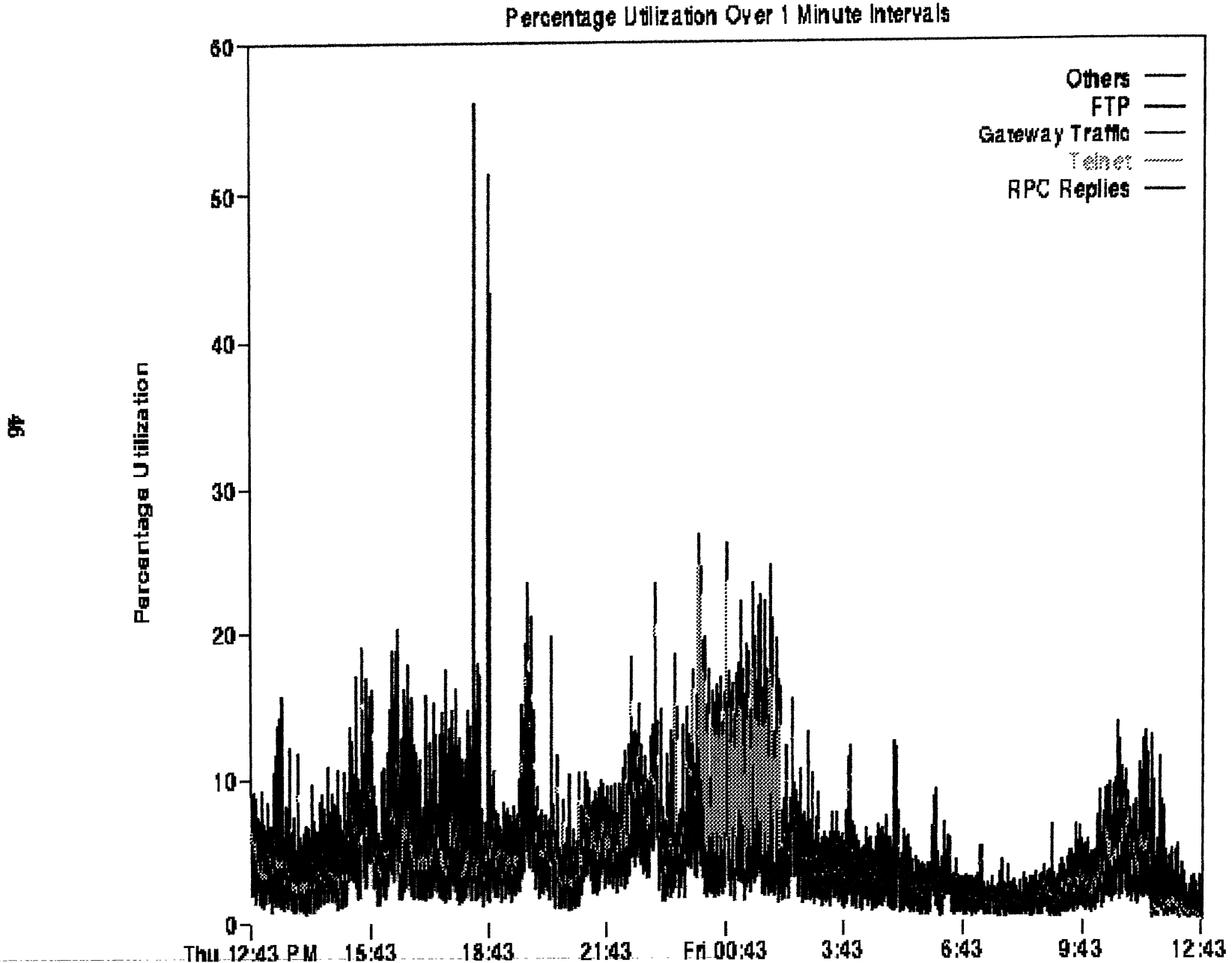


Figure 19: Percentage Utilization

Figure 20 shows some components of the overall traffic. Each spike represents the total utilization and colors show the contribution due to individual protocols.


Percentage Utilization Over 1 Minute Intervals
Others
FTP
Gateway Traffic
Telnet
RPC Replies
Percentage Utilization
60
50
40
30
20
10
0
Thu 12:43 PM
15:43
18:43
21:43
Fri 00:43
3:43
6:43
9:43
12:43

In the next few graphs we look at few dominant applications.

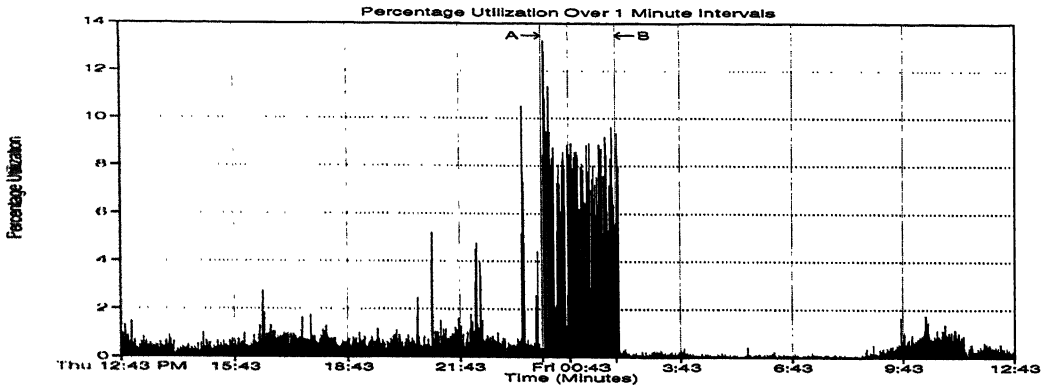


Figure 21: Ethernet Utilization Over a Period Of 24 Hours (Telnet)

The telnet protocol (figure 21) has a mean utilization of 1.04%. Except for a few peak hours, the utilization is most of the time below 1%. During the peak utilization period, marked in the figure, the average utilization is 6.54%.

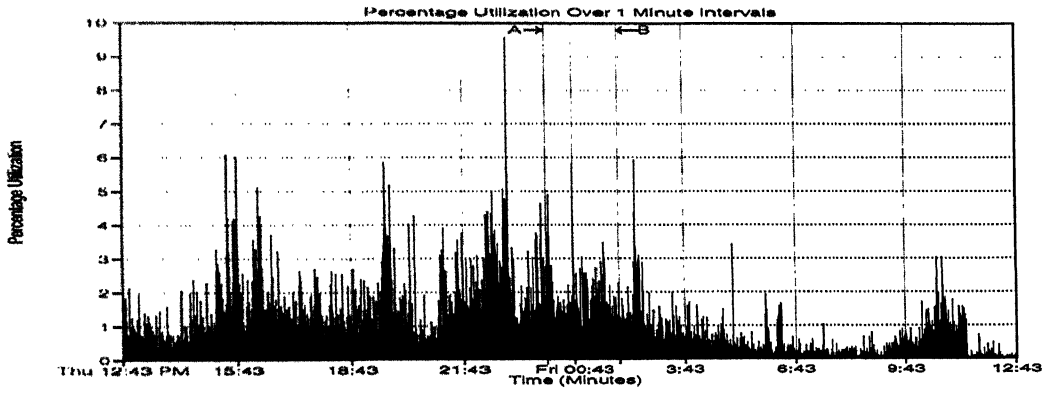


Figure 22: Ethernet Utilization Over a Period Of 24 Hours (NFS Calls)

NFS calls (figure 22) constitute a sizeable portion of the total utilization. The mean utilization by NFS calls is 1.24%. Since the server file systems, on which the user home directories are situated are mounted on the workstations and Linux machines, the NFS call traffic is generated during file writes and other calls whenever

the remote file systems are accessed. During the peak utilization period, marked in the figure, the average utilization is 1.85%.

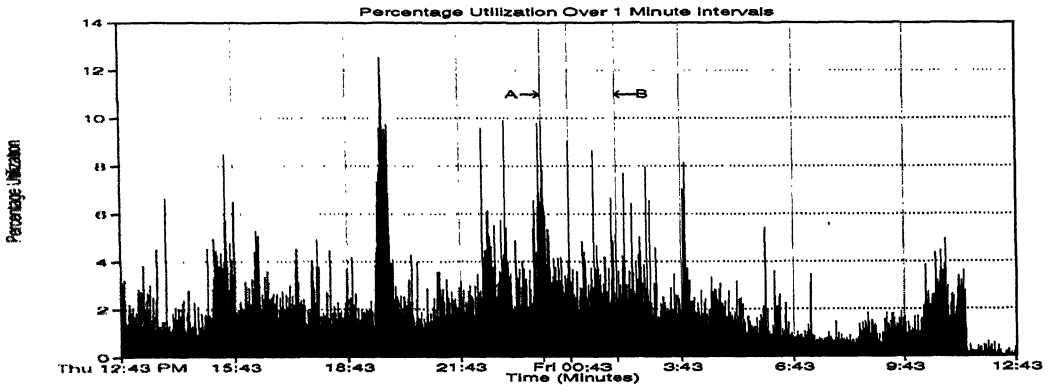


Figure 23: Ethernet Utilization Over a Period Of 24 Hours (RPC Replies)

RPC replies (figure 23) consists mainly of NFS and YP (NIS) replies. A lot of traffic is due to file read and lookup operations. The mean utilization due to RPC replies is 2.23% and during the peak activity period, the mean utilization is 3.44%.

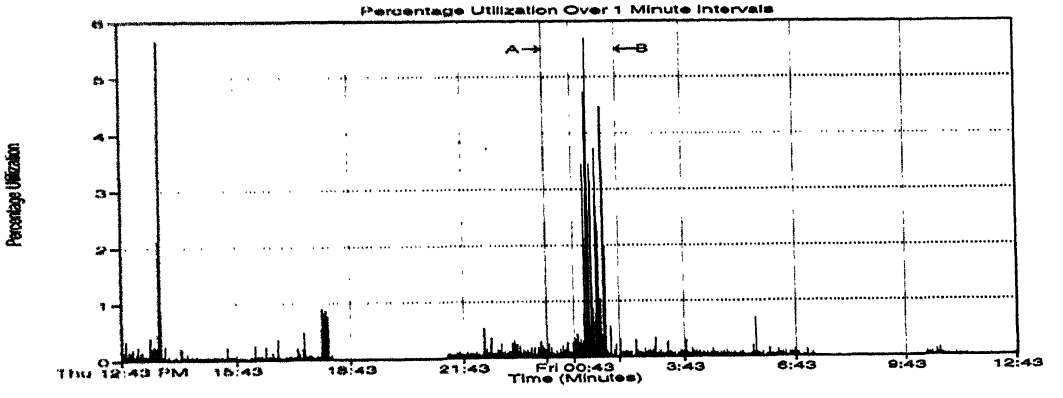


Figure 24: Ethernet Utilization Over a Period Of 24 Hours (Gateway Traffic)

Figure 24 shows the utilization due to traffic from/to *yamuna:3128*, the IITK proxy server. The traffic is very low because the bottleneck is the link going out of IITK. There is no traffic between between 18:00-21:00 hrs and 7:00-10:00 hrs when yamuna does not accept any requests. There is a sudden increase in gateway

traffic at 0:00 midnight after which time running browsers is officially allowed on CC machines. The average utilization for the peak period is 0.66%.

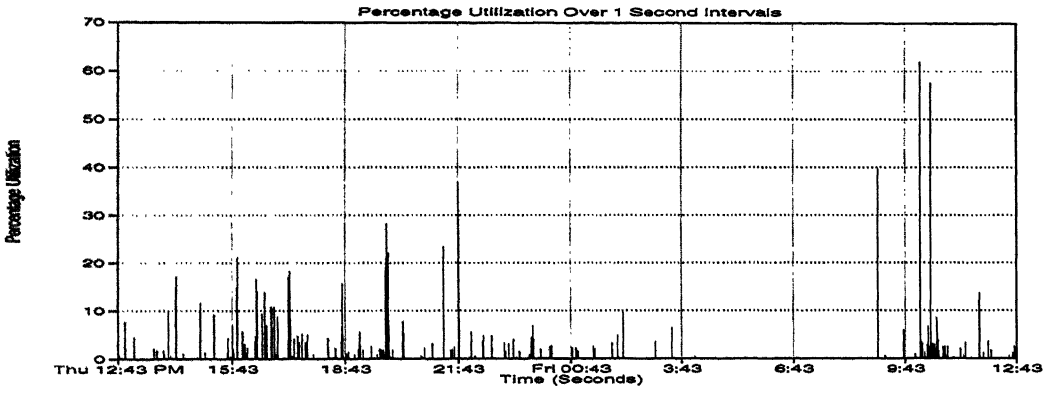


Figure 25: Ethernet Utilization Over a Period Of 24 Hours (FTP)

The FTP traffic (figure 25) is highly bursty. During some periods the utilization is as high as 60% but the mean utilization over a day is only 0.05%. The peaks in overall Ethernet utilization are usually due to FTP bursts.

4.4 Other Observations

The network response time was measured through the Echo Server described in Chapter 3. The response time was checked against the network load. There was absolutely no dependence of the response time on the network load. The response time depended on factors other than the network load with the system load being the main contributing factor in the large response times observed on the CC systems.

There were quite a few IP clashes found during the monitoring. Also, a large IP-Ethernet mapping database has been created which can be very useful in system administration.

Chapter 5 discusses the conclusions drawn from the study and possibility of future work.

Chapter 5

Conclusion and Future Work

Indian Institute Of Technology has one of the biggest networks (amongst educational institutions) in India. Unfortunately, there is no software which can monitor and manage heterogeneous networks. This study is an attempt to monitor and analyze traffic on the IIT Kanpur network. This chapter discusses some conclusions drawn from the study and the directions for future work.

5.1 Conclusion

The monitoring study on the IITK CC Network brought out some new points and confirmed other already reported observations. It has become extremely clear from the observations that the network is lightly loaded and the existing Ethernet is good enough for the kind of applications presently in use. On the CC Network there was no correlation between the response time and the network load measured through the Echo server. The delayed response time problem on the CC network is due to either system load or due to abrupt hanging of the Ethernet switch at the Computer Center. It has been confirmed through another independent study [Kum97] on the Computer Center machines that some of the machines are indeed heavily loaded. A few IP clashes were reported and a large IP-Ethernet mapping database was created during the monitoring.

5.2 Suggestions

The cable connections at the computer center should also be checked properly. We suspect loose cable connection are resulting in large packet losses. Also, the Ethernet switch has to be properly configured. Currently, machines at the computer center are using 'hard' mount which results in a busy wait if the NFS server is unavailable for some reason. The mountings should be either 'soft' or *automounter* should be used. The configuration of the CC network may be changed so that there is better utilization of the available infrastructure.

5.3 Future Work

Some more work needs to be done to improve the monitoring tool. A good user friendly Graphical User Interface (GUI) needs to be built for the monitoring tool. Some SNMP support can also be added as devices on the network start supporting SNMP. The monitoring tool can be made distributed in nature with different servers running on separate segments and exchanging information with each other to build an overall picture of the network traffic.

More analysis of the data collected may be done to get a better idea about the state of the network. For example Time-Series analysis can be used to find out patterns in the network traffic.

Bibliography

[Bla92] Matt Blaze. Nfs tracing by passive network monitoring. *USENIX conference*, pages 333–344, Jan 1992.

[CM] Dave Curry and Jeff Mogul. URL. *ftp://coast.cs.purdue.edu/pub/tools/unix/nfswatch/nfswatch4.2.tar.gz.*

[CN] Inc. Cinco Networks. URL. *http://www.nomadix.com/SampleMWeb/cin3.htm.*

[Com95] Douglas E. Comer. *INTERNETWORKING WITH TCP/IP*, volume I. PHI, third edition, 1995.

[Cor] 3Com Corporation. URL. *http://www.3com.com/products/dsheets/400294.html.*

[Cor94] Digital Equipment Corporation. *Unix Programming Manual.* Feb 1994.

[Dem] Laurent Demailly. URL. *http://hplyot.obspm.fr/~dl/icmpinfo.html.*

[DEMK75] Dale P. Goodspeed David E. Morgan, Walter Banks and Richard Kolanko. A computer network monitoring system. *IEEE Transactions on Software Engineering*, SE-1(3):299–311, Sep 1975.

[DRBK88] J. C. Mogul D. R. Boggs and C. A. Kent. Measured capacity of an ethernet: Myths and reality. *Proceedings of SIGCOMM*, pages 222–234, 1988.

[Fei95] Sidnie Feit. *SNMP: A Guide to Network Management.* McGraw-Hill, Inc., 1995.

[GC] UC Irvine Academic Computing Guy Cardwell. URL. *http://www.intac.com/man/sunos/rpc.etherd.8c.html.*

[GNU93] GNU. *Linux Programmer's Manual.* April 1993.

[Groa] Netman Development Group. URL. *http://www.cs.curtin.edu.au/~netman.*

[Grob] Network Research Group. URL. *ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.*

[GT88] T. A. Gonsalves and F. A. Tobagi. On the performance effects of stations and access protocol parameters in ethernet networks. *IEEE Transactions on Communications*, 36(4):441–449, April 1988.

[Gus90] Riccardo Gusella. A measurement study of diskless workstation traffic on an ethernet. *IEEE Transactions on Communications*, 38(9):1557–1568, September 1990.

[HO86] Joseph L. Hammond and Peter J.P. O'Reilly. *Performance Analysis Of Local Computer Networks.* Addition-Wesley Publishing Company, 1986.

[Hom] The Public Netperf Homepage. URL. *http://www.cup.hp.com/netperf/NetperfPage.html.*

[Jai91] Raj Jain. *The Art Of Computer Systems Performance Analysis.* John Wiley and Sons, INC, New York, 1991.

[JM] Van Jacobson and Steve McCanne. URL. *ftp://ftp.ee.lbl.gov/netload.tar.Z.*

[JR86] Raj Jain and Shawn A. Routhier. Packet trains - measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):986–995, Jan 1986.

[KO77] Leonard Kleinrock and Holger Opderbeck. Throghput in the ARPANET-protocols and measurement. *IEEE Transactions on Communications*, COM-25(1):95–104, Jan 1977.

[Kum97] Atul Kumar. A performance evaluation and benchmarking tool. Master's thesis, Indian Institute Of Technology Kanpur, India, Dec 1997.

[Ltd] NetWin Ltd. URL. *http://netwinsite.com/watchdog1.htm.*

[Man93] BSD Manpage. *Linux Programmer's Manual.* November 1993.

[Mic89] Sun Microsystems. *Unix Programming Manual.* march 1989.

[Rao97] P. V. Nageswara Rao. NETMON: A network monitoring tool. Master's thesis, Indian Institute Of Technology Kanpur, India, 1997.

[SF] Mike Shulze and Craig Farell. URL. *ftp://ftp.cs.curtin.edu.au.*

[SH80] John F. Shoch and Jon A. Hupp. Measured performance of an ethernet local network. *Communications of the ACM*, 23(12):711–721, December 1980.

[SMJa] Craig Leres Steve McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. *ftp://ftp.ee.lbl.gov/papers/bpf-usenix93.ps.Z.*

[SMJb] Craig Leres Steve McCanne and Van Jacobson. URL. *ftp://ftp.ee.lbl .gov/libpcap.tar.Z.*

[Ste90] Richard W. Stevens. *Unix Network Programming.* PHI, 1990.